Workgroup: HTTP
Internet-Draft:
draft-ietf-httpbis-digest-headers-08
Obsoletes: 3230 (if approved)
Published: 21 March 2022
Intended Status: Standards Track
Expires: 22 September 2022
Authors: R. Polli, Team Digitale, Italian Government
L. Pardue, Cloudflare

Digest Fields

Abstract

This document defines HTTP fields that support integrity digests. The Repr-Digest field can be used for the integrity of HTTP representations. The Content-Digest field can be used for the integrity of HTTP message content. Want-Repr-Digest and Want-Content-Digest can be used to indicate a sender's interest and preferences for receiving the respective Integrity fields.

This document obsoletes RFC 3230 and the Digest and Want-Digest HTTP fields.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-httpbis-digest-headers/.

Discussion of this document takes place on the HTTP Working Group mailing list (mailto:ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/. Working Group information can be found at https://httpwg.org/.

Source for this draft and an issue tracker can be found at https://github.com/httpwg/http-extensions/labels/digest-headers.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

HTTP does not define the means to protect the data integrity of representations or content. When HTTP messages are transferred between endpoints, lower layer features or properties such as TCP checksums or TLS records [RFC2818] can provide some integrity protection. However, transport-oriented integrity provides a limited utility because it is opaque to the application layer and only covers the extent of a single connection. HTTP messages often travel over a chain of separate connections, in between connections there is a possibility for unintended or malicious data corruption. An HTTP integrity mechanism can provide the means for endpoints, or applications using HTTP, to detect data corruption and make a choice about how to act on it. An example use case is to aid fault detection and diagnosis across system boundaries.

This document defines two digest integrity mechanisms for HTTP. First, representation data integrity, which acts on representation data (Section 3.2 of [SEMANTICS]). This supports advanced use cases such as validating the integrity of a resource that was reconstructed from parts retrieved using multiple requests or connections. Second, content integrity, which acts on conveyed content (Section 6.4 of [SEMANTICS]).

This document obsoletes RFC 3230 and therefore the Digest and Want-Digest HTTP fields; see Section 1.3.

1.1. Document Structure

This document is structured as follows:

* Section 2 defines the Repr-Digest request and response header and trailer field,
* Section 3 defines the Content-Digest request and response header and trailer field,
* Section 4 defines the Want-Repr-Digest and Want-Content-Digest request and response header and trailer field,
* Section 5 describes algorithms and their relation to the fields defined in this document,
* Section 2.1 details computing representation digests,
* Appendix B and Appendix C provide examples of using Repr-Digest and Want-Repr-Digest.

1.2. Concept Overview

The HTTP fields defined in this document can be used for HTTP integrity. Senders choose a hashing algorithm and calculate a digest from an input related to the HTTP message, the algorithm identifier and digest are transmitted in an HTTP field. Receivers can validate the digest for integrity purposes. Hashing algorithms are registered in the "Hash Algorithms for HTTP Digest Fields" (see Section 5).

Selecting the data on which digests are calculated depends on the use case of HTTP messages. This document provides different headers for HTTP representation data and HTTP content.

This document defines the Repr-Digest request and response header and trailer field (Section 2) that contains a digest value computed by applying a hashing algorithm to "selected representation data" (Section 3.2 of [SEMANTICS]). Basing Repr-Digest on the selected representation makes it straightforward to apply it to use-cases where the transferred data requires some sort of manipulation to be considered a representation or conveys a partial representation of a resource, such as Range Requests (see Section 14.2 of [SEMANTICS]).

There are use-cases where a simple digest of the HTTP content bytes is required. The Content-Digest request and response header and trailer field is defined to support digests of content (Section 3.2 of [SEMANTICS]); see Section 3.

Repr-Digest and Content-Digest support hashing algorithm agility. The Want-Repr-Digest and Want-Content-Digest fields allows endpoints to express interest in Repr-Digest and Content-Digest respectively, and preference of algorithms in either.

Repr-Digest and Content-Digest are collectively termed Integrity fields. Want-Repr-Digest and Want-Content-Digestare collectively termed Integrity preference fields.

Integrity fields are tied to the Content-Encoding and Content-Type header fields. Therefore, a given resource may have multiple different digest values when transferred with HTTP.

Integrity fields do not provide integrity for HTTP messages or fields. However, they can be combined with other mechanisms that protect metadata, such as digital signatures, in order to protect the phases of an HTTP exchange in whole or in part.

This specification does not define means for authentication, authorization or privacy.

1.3. Obsoleting RFC 3230

[RFC3230] defined the Digest and Want-Digest HTTP fields for HTTP integrity. It also coined the term "instance" and "instance manipulation" in order to explain concepts that are now more universally defined, and implemented, as HTTP semantics such as "selected representation data" (Section 3.2 of [SEMANTICS]).

Experience has shown that implementations of [RFC3230] have interpreted the meaning of "instance" inconsistently, leading to interoperability issues. The most common mistake being the calculation of the digest using (what we now call) message content, rather than using (what we now call) representation data as was originally intended. Interestingly, time has also shown that a digest of message content can be beneficial for some use cases. So it is difficult to detect if non-conformance to [RFC3230] is intentional or unintentional.

In order to address potential inconsistencies and ambiguity across implementations of Digest and Want-Digest, this document obsoletes

[RFC3230]. The Integrity fields (Section 2 and Section 3) and Integrity preference fields (Section 4) defined in this document are better aligned with current HTTP semantics and have names that more clearly articulate the intended usages.

1.4. Notational Conventions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the Augmented BNF defined in [RFC5234] and updated by [RFC7405].

This document uses the Boolean, Byte Sequence, Dictionary, Integer and List types from [STRUCTURED-FIELDS] along with the sf-dictionary and sf-list ABNF rules.

The definitions "representation", "selected representation", "representation data", "representation metadata", "user agent" and "content" in this document are to be interpreted as described in [SEMANTICS].

Hashing algorithm names respect the casing used in their definition document (e.g. SHA-1, CRC32c) whereas hashing algorithm keys are quoted (e.g. "sha", "crc32c").

The term "checksum" describes the output of the application of an algorithm to a sequence of bytes, whereas "digest" is only used in relation to the value contained in the fields.

Integrity fields: collective term for Repr-Digest and Content-Digest

Integrity preference fields: collective term for Want-Repr-Digest and Want-Content-Digest

2. The Repr-Digest Field

The Repr-Digest HTTP field can be used in requests and responses to communicate digests that are calculated using a hashing algorithm applied to the entire "selected representation data" (see Section 8.1 of [SEMANTICS]).

Representations take into account the effect of the HTTP semantics on messages. For example, the content can be affected by Range Requests or methods such as HEAD, while the way the content is transferred "on the wire" is dependent on other transformations (e.g. transfer codings for HTTP/1.1 - see Section 6.1 of [HTTP11]).

To help illustrate HTTP representation concepts, several examples are provided in Appendix A.

When a message has no "representation data" it is still possible to assert that no "representation data" was sent by computing the digest on an empty string (see Section 6.3).

Repr-Digest is a Structured Fields Dictionary (see Section 3.2 of [STRUCTURED-FIELDS]) where:

* keys convey the hashing algorithm (see Section 5) used to compute the digest;

* values **MUST** be of type Byte Sequence, which contain the output of the digest calculation.

```
Repr-Digest   = sf-dictionary
```

For example:

```
NOTE: '\' line wrapping per RFC 8792

Repr-Digest: \
  sha-512=:WZDPaVn/7XgHaAy8pmojAkGWoRx2UFChF41A2svX+TaPm+AbwAgBWnrI\
  iYllu7BNNyealdVLvRwEmTHWXvJwew==:
```

The Dictionary type can be used, for example, to attach multiple digests calculated using different hashing algorithms in order to support a population of endpoints with different or evolving capabilities. Such an approach could support transitions away from weaker algorithms (see Section 6.6).

```
NOTE: '\' line wrapping per RFC 8792

Repr-Digest: \
  sha-256=:4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=:,\
  sha-512=:WZDPaVn/7XgHaAy8pmojAkGWoRx2UFChF41A2svX+TaPm+AbwAgBWnrI\
  iYllu7BNNyealdVLvRwEmTHWXvJwew==:
```

A recipient **MAY** ignore any or all digests. This allows the recipient to choose which hashing algorithm(s) to use for validation instead of verifying every digest.

A sender **MAY** send a digest without knowing whether the recipient supports a given hashing algorithm, or even knowing that the recipient will ignore it.

Repr-Digest can be sent in a trailer section. In this case, Repr-Digest **MAY** be merged into the header section; see Section 6.5.1 of [SEMANTICS].

2.1. Using Repr-Digest in State-Changing Requests

When the representation enclosed in a state-changing request does not describe the target resource, the representation digest **MUST** be computed on the representation data. This is the only possible choice because representation digest requires complete representation metadata (see Section 2).

In responses,

* if the representation describes the status of the request, Repr-Digest **MUST** be computed on the enclosed representation (see Appendix B.8);

* if there is a referenced resource Repr-Digest **MUST** be computed on the selected representation of the referenced resource even if that is different from the target resource. That might or might not result in computing Repr-Digest on the enclosed representation.

The latter case is done according to the HTTP semantics of the given method, for example using the Content-Location header field (see Section 8.7 of [SEMANTICS]). In contrast, the Location header field does not affect Repr-Digest because it is not representation metadata.

For example, in PATCH requests, the representation digest will be computed on the patch document because the representation metadata refers to the patch document and not to the target resource (see Section 2 of [PATCH]). In responses, instead, the representation digest will be computed on the selected representation of the patched resource.

2.2. Repr-Digest and Content-Location in Responses

When a state-changing method returns the Content-Location header field, the enclosed representation refers to the resource identified by its value and Repr-Digest is computed accordingly. An example is given in Appendix B.7.

3. The Content-Digest Field

The Content-Digest HTTP field can be used in requests and responses to communicate digests that are calculated using a hashing algorithm applied to the actual message content (see Section 6.4 of [SEMANTICS]). It is a Structured Fields Dictionary (see Section 3.2 of [STRUCTURED-FIELDS]) where:

* keys convey the hashing algorithm (see Section 5) used to compute the digest;
* values **MUST** be Byte Sequences (Section 3.3.5 of [STRUCTURED-FIELDS]) containing the output of the digest calculation.

```
Content-Digest   = sf-dictionary
```

For example:

```
NOTE: '\' line wrapping per RFC 8792

Content-Digest: \
  sha-512=:WZDPaVn/7XgHaAy8pmojAkGWoRx2UFChF41A2svX+TaPm+AbwAgBWnrI\
  iYllu7BNNyealdVLvRwEmTHWXvJwew==:
```

The Dictionary type can be used, for example, to attach multiple digests calculated using different hashing algorithms in order to support a population of endpoints with different or evolving capabilities. Such an approach could support transitions away from weaker algorithms (see Section 6.6).

```
NOTE: '\' line wrapping per RFC 8792

Repr-Digest: \
  sha-256=:4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=:,\
  sha-512=:WZDPaVn/7XgHaAy8pmojAkGWoRx2UFChF41A2svX+TaPm+AbwAgBWnrI\
  iYllu7BNNyealdVLvRwEmTHWXvJwew==:
```

A recipient **MAY** ignore any or all digests. This allows the recipient to choose which hashing algorithm(s) to use for validation instead of verifying every digest.

A sender **MAY** send a digest without knowing whether the recipient supports a given hashing algorithm, or even knowing that the recipient will ignore it.

Content-Digest can be sent in a trailer section. In this case, Content-Digest **MAY** be merged into the header section; see Section 6.5.1 of [SEMANTICS].

4. Integrity preference fields

Senders can indicate their interest in Integrity fields and hashing algorithm preferences using the Want-Repr-Digest or Want-Content-Digest fields. These can be used in both requests and responses.

Want-Repr-Digest indicates the sender's desire to receive a representation digest on messages associated with the request URI and representation metadata, using the Repr-Digest field.

Want-Content-Digest indicates the sender's desire to receive a content digest on messages associated with the request URI and representation metadata, using the Content-Digest field.

Want-Repr-Digest and Want-Content-Digest are Structured Fields Dictionary (see Section 3.2 of [STRUCTURED-FIELDS]) where:

* keys convey the hashing algorithm (see Section 5);

* values **MUST** be of type Integer (Section 3.3.1 of [STRUCTURED-FIELDS]) in the range 0 to 10 inclusive. 1 is the least preferred, 10 is the most preferred, and a value of 0 means "not acceptable". Values convey an ascending, relative, weighted preference.

```
Want-Repr-Digest = sf-dictionary
Want-Content-Digest = sf-dictionary
```

Examples:

```
Want-Repr-Digest: sha-256=1
Want-Repr-Digest: sha-512=3, sha-256=10, unixsum=0
Want-Content-Digest: sha-256=1
Want-Content-Digest: sha-512=3, sha-256=10, unixsum=0
```

5. Hash Algorithms for HTTP Digest Fields Registry

The "Hash Algorithms for HTTP Digest Fields", maintained by IANA at https://www.iana.org/assignments/http-dig-alg/, registers algorithms for use with the Integrity and Integrity preference fields defined in this document.

This registry uses the Specification Required policy (Section 4.6 of [RFC8126]).

Registrations **MUST** include the following fields:

* Algorithm Key: the Structured Fields key value used in Repr-Digest, Content-Digest, Want-Repr-Digest or Want-Content-Digest field Dictionary member keys

* Status: the status of the algorithm. Use "standard" for standardized algorithms without known problems; "experimental" or some other appropriate value

  - e.g. according to the type and status of the primary document in which the algorithm is defined; "insecure" when the algorithm is insecure; "reserved" when the algorithm references a reserved token value

* Description: a short description of the algorithm

* Reference(s): a set of pointers to the primary documents defining the algorithm and key

Insecure hashing algorithms **MAY** be used to preserve integrity against corruption, but **MUST NOT** be used in a potentially adversarial setting; for example, when signing Integrity fields' values for authenticity.

The entries in Table 1 are registered by this document.

Algorithm Key	**Status**	**Description**	**Reference(s)**
sha-512	standard	The SHA-512 algorithm.	[RFC6234], [RFC4648], this document.
sha-256	standard	The SHA-256 algorithm.	[RFC6234], [RFC4648], this document.
md5	insecure	The MD5 algorithm. It is vulnerable to collision attacks; see [NO-MD5] and [CMU-836068]	[RFC1321], [RFC4648], this document.
sha	insecure	The SHA-1 algorithm. It is vulnerable to collision attacks; see [NO-SHA] and [IACR-2020-014]	[RFC3174], [RFC4648], [RFC6234] this document.

Algorithm Key	Status	Description	Reference(s)
unixsum	insecure	The algorithm used by the UNIX "sum" command.	[RFC4648], [RFC6234], [UNIX], this document.
unixcksum	insecure	The algorithm used by the UNIX "cksum" command.	[RFC4648], [RFC6234], [UNIX], this document.
adler	insecure	The ADLER32 algorithm.	[RFC1950], this document.
crc32c	insecure	The CRC32c algorithm.	[RFC4960] appendix B, this document.

Table 1: Initial Hash Algorithms

6. Security Considerations

6.1. HTTP Messages Are Not Protected In Full

This document specifies a data integrity mechanism that protects HTTP "representation data" or content, but not HTTP header and trailer fields, from certain kinds of corruption.

Integrity fields are not intended to be a general protection against malicious tampering with HTTP messages. This can be achieved by combining it with other approaches such as transport-layer security or digital signatures.

6.2. End-to-End Integrity

Integrity fields can help detect "representation data" or content modification due to implementation errors, undesired "transforming proxies" (see Section 7.7 of [SEMANTICS]) or other actions as the data passes across multiple hops or system boundaries. Even a simple mechanism for end-to-end "representation data" integrity is valuable because a user agent can validate that resource retrieval succeeded before handing off to a HTML parser, video player etc. for parsing.

Note that using these mechanisms alone does not provide end-to-end integrity of HTTP messages over multiple hops, since metadata could be manipulated at any stage. Methods to protect metadata are discussed in Section 6.3.

6.3. Usage in Signatures

Digital signatures are widely used together with checksums to provide the certain identification of the origin of a message [NIST800-32]. Such signatures can protect one or more HTTP fields and there are additional considerations when Integrity fields are included in this set.

Digests explicitly depend on the "representation metadata" (e.g. the values of Content-Type, Content-Encoding etc). A signature that protects Integrity fields but not other "representation metadata" can expose the communication to tampering. For example, an actor could manipulate the Content-Type field-value and cause a digest validation failure at the recipient, preventing the application from accessing the representation. Such an attack consumes the resources of both endpoints. See also Section 2.2.

Signatures are likely to be deemed an adversarial setting when applying Integrity fields; see Section 5. Using signatures to protect the checksum of an empty representation allows receiving endpoints to detect if an eventual payload has been stripped or added.

Any mangling of Integrity fields, including digests' de-duplication or combining different field values (see Section 5.2 of [SEMANTICS]) might affect signature validation.

6.4. Usage in Trailer Fields

Before sending Integrity fields in a trailer section, the sender should consider that intermediaries are explicitly allowed to drop any trailer (see Section 6.5.2 of [SEMANTICS]).

When Integrity fields are used in a trailer section, the field-values are received after the content. Eager processing of content before the trailer section prevents digest validation, possibly leading to processing of invalid data.

Not every hashing algorithm is suitable for use in the trailer section, some may require to pre-process the whole payload before sending a message (e.g. see [I-D.thomson-http-mice]).

6.5. Usage with Encryption

The checksum of an encrypted payload can change between different messages depending on the encryption algorithm used; in those cases its value could not be used to provide a proof of integrity "at rest" unless the whole (e.g. encoded) content is persisted.

6.6. Algorithm Agility

The security properties of hashing algorithms are not fixed. Algorithm Agility (see [RFC7696]) is achieved by providing implementations with flexibility to choose hashing algorithms from the IANA Hash Algorithms for HTTP Digest Fields registry; see Section 7.2.

The "standard" algorithms listed in this document are suitable for many purposes, including adversarial situations where hash functions might need to provide resistance to collision, first-preimage and second-preimage attacks. Algorithms listed as "insecure" either provide none of these properties, or are known to be weak (see [NO-MD5] and [NO-SHA]).

For adversarial situations, which of the "standard" algorithms are acceptable will depend on the level of protection the circumstances demand. As there is no negotiation, endpoints that depend on a digest for security will be vulnerable to attacks on the weakest algorithm they are willing to accept.

Transition from weak algorithms is supported by negotiation of hashing algorithm using Want-Repr-Digest or Want-Content-Digest (see Section 4) or by sending multiple digests from which the receiver chooses. Endpoints are advised that sending multiple values consumes resources, which may be wasted if the receiver ignores them (see Section 2).

While algorithm agility allows the migration to stronger algorithms it does not prevent the use of weaker algorithms. Integrity fields do not provide any mitigiations for downgrade or substitution attacks (see Section 1 of [RFC6211]) of the hashing algorithm. To protect against such attacks, endpoints could restrict their set of supported algorithms to stronger ones and protect the fields value by using TLS and/or digital signatures.

6.7. Resource exhaustion

Integrity fields validation consumes computational resources. In order to avoid resource exhaustion, implementations can restrict validation of the algorithm types, number of validations, or the size of content.

7. IANA Considerations

7.1. HTTP Field Name Registration

IANA is asked to update the "Hypertext Transfer Protocol (HTTP) Field Name Registry" registry ([SEMANTICS]) according to the table below:

Field Name	Status	Reference
Repr-Digest	permanent	Section 2 of this document
Content-Digest	permanent	Section 3 of this document
Want-Repr-Digest	permanent	Section 4 of this document
Want-Content-Digest	permanent	Section 4 of this document

Field Name	Status	Reference
Digest	obsoleted	[RFC3230], Section 1.3 of this document
Want-Digest	obsoleted	[RFC3230], Section 1.3 of this document

Table 2

7.2. Establish the Hash Algorithms for HTTP Digest Fields Registry

This memo sets this specification to be the establishing document for the Hash Algorithms for HTTP Digest Fields registry defined in Section 5.

IANA is asked to initialize the registry with the entries in Table 1.

8. References

8.1. Normative References

[CMU-836068] Carnagie Mellon University, Software Engineering Institute, "MD5 Vulnerable to collision attacks", 31 December 2008, <https://www.kb.cert.org/vuls/id/836068/>.

[IACR-2020-014] Leurent, G. and T. Peyrin, "SHA-1 is a Shambles", 5 January 2020, <https://eprint.iacr.org/2020/014.pdf>.

[NIST800-32] National Institute of Standards and Technology, U.S. Department of Commerce, "Introduction to Public Key Technology and the Federal PKI Infrastructure", February 2001, <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-32.pdf>.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992, <https://www.rfc-editor.org/rfc/rfc1321>.

[RFC1950] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, DOI 10.17487/RFC1950, May 1996, <https://www.rfc-editor.org/rfc/rfc1950>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3174] Eastlake 3rd, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, DOI 10.17487/RFC3174, September 2001, <https://www.rfc-editor.org/rfc/rfc3174>.

[RFC3230]
Mogul, J. and A. Van Hoff, "Instance Digests in HTTP", RFC 3230, DOI 10.17487/RFC3230, January 2002, <https://www.rfc-editor.org/rfc/rfc3230>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/rfc/rfc4960>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/rfc/rfc5234>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/rfc/rfc6234>.

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <https://www.rfc-editor.org/rfc/rfc7405>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[SEMANTICS] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>.

[STRUCTURED-FIELDS] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/rfc/rfc8941>.

[UNIX] The Open Group, "The Single UNIX Specification, Version 2 - 6 Vol Set for UNIX 98", February 1997.

8.2. Informative References

[HTTP11]
Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP/1.1", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-19, 12 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-messaging-19>.

[I-D.thomson-http-mice] Thomson, M. and J. Yasskin, "Merkle Integrity Content Encoding", Work in Progress, Internet-Draft, draft-thomson-http-mice-03, 13 August 2018, <https://datatracker.ietf.org/doc/html/draft-thomson-http-mice-03>.

[NO-MD5] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <https://www.rfc-editor.org/rfc/rfc6151>.

[NO-SHA] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <https://www.rfc-editor.org/rfc/rfc6194>.

[PATCH] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <https://www.rfc-editor.org/rfc/rfc5789>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/rfc/rfc2818>.

[RFC6211] Schaad, J., "Cryptographic Message Syntax (CMS) Algorithm Identifier Protection Attribute", RFC 6211, DOI 10.17487/RFC6211, April 2011, <https://www.rfc-editor.org/rfc/rfc6211>.

[RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <https://www.rfc-editor.org/rfc/rfc7396>.

[RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <https://www.rfc-editor.org/rfc/rfc7696>.

[RFC7807] Nottingham, M. and E. Wilde, "Problem Details for HTTP APIs", RFC 7807, DOI 10.17487/RFC7807, March 2016, <https://www.rfc-editor.org/rfc/rfc7807>.

Appendix A. Resource Representation and Representation Data

The following examples show how representation metadata, payload transformations and method impacts on the message and content. When the content contains non-printable characters (e.g. when it is compressed) it is shown as a Base64-encoded string.

```
PUT /entries/1234 HTTP/1.1
Host: foo.example
Content-Type: application/json

{"hello": "world"}
```

Figure 1: Request containing a JSON object without any content coding

```
PUT /entries/1234 HTTP/1.1
Host: foo.example
Content-Type: application/json
Content-Encoding: gzip

H4sIAItWyFwC/6tWSlSyUlAypANQqgUAREcqfG0AAAA=
```

Figure 2: Request containing a gzip-encoded JSON object

Now the same content conveys a malformed JSON object, because the request does not indicate a content coding.

```
PUT /entries/1234 HTTP/1.1
Host: foo.example
Content-Type: application/json

H4sIAItWyFwC/6tWSlSyUlAypANQqgUAREcqfG0AAAA=
```

Figure 3: Request containing malformed JSON

A Range-Request alters the content, conveying a partial representation.

```
GET /entries/1234 HTTP/1.1
Host: foo.example
Range: bytes=1-7
```

Figure 4: Request for partial content

```
HTTP/1.1 206 Partial Content
Content-Encoding: gzip
Content-Type: application/json
Content-Range: bytes 1-7/18

iwgAla3RXA==
```

Figure 5: Partial response from a gzip-encoded representation

The method can also alter the content. For example, the response to a HEAD request does not carry content.

```
HEAD /entries/1234 HTTP/1.1
Host: foo.example
Accept: application/json
Accept-Encoding: gzip
```

Figure 6: HEAD request

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: gzip
```

Figure 7: Response to HEAD request (empty content)

Finally, the semantics of an HTTP response might decouple the effective request URI from the enclosed representation. In the example response below, the Content-Location header field indicates that the enclosed representation refers to the resource available at /authors/123, even though the request is directed to /authors/.

```
POST /authors/ HTTP/1.1
Host: foo.example
Accept: application/json
Content-Type: application/json

{"author": "Camilleri"}
```

Figure 8: POST request

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Location: /authors/123
Location: /authors/123

{"id": "123", "author": "Camilleri"}
```

Figure 9: Response with Content-Location header

Appendix B. Examples of Unsolicited Digest

The following examples demonstrate interactions where a server responds with a Repr-Digest or Content-Digest fields even though the client did not solicit one using Want-Repr-Digest or Want-Content-Digest.

Some examples include JSON objects in the content. For presentation purposes, objects that fit completely within the line-length limits are presented on a single line using compact notation with no leading space. Objects that would exceed line-length limits are presented across multiple lines (one line per key-value pair) with 2 spaced of leading indentation.

Checksum mechanisms defined in this document are media-type agnostic and do not provide canonicalization algorithms for specific formats. Examples are calculated inclusive of any space. While examples can include both fields, Repr-Digest and Content-Digest can be returned independently.

B.1. Server Returns Full Representation Data

In this example, the message content conveys complete representation data. This means that in the response, Repr-Digest and Content-Digest are both computed over the JSON object {"hello": "world"}, and thus have the same value.

```
GET /items/123 HTTP/1.1
Host: foo.example
```

Figure 10: GET request for an item

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 200 OK
Content-Type: application/json
Repr-Digest: \
  sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:
Content-Digest: \
  sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:

{"hello": "world"}
```

Figure 11: Response with identical Repr-Digest and Content-Digest

B.2. Server Returns No Representation Data

In this example, a HEAD request is used to retrieve the checksum of a resource.

The response Repr-Digest field-value is calculated over the JSON object {"hello": "world"}, which is not shown because there is no payload data. Content-Digest is computed on empty content.

```
HEAD /items/123 HTTP/1.1
Host: foo.example
```

Figure 12: HEAD request for an item

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 200 OK
Content-Type: application/json
Repr-Digest: \
  sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:
Content-Digest: \
  sha-256=:47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=:
```

Figure 13: Response with both Content-Digest and Digest; empty content

B.3. Server Returns Partial Representation Data

In this example, the client makes a range request and the server responds with partial content.

```
GET /items/123 HTTP/1.1
Host: foo.example
Range: bytes=1-7
```

Figure 14: Request for partial content

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 206 Partial Content
Content-Type: application/json
Content-Range: bytes 1-7/18
Repr-Digest: \
  sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:
Content-Digest: \
  sha-256=:Wqdirjg/u3J688ejbUlApbjECpiUUtIwT8lY/z81Tno=:

"hello"
```

Figure 15: Partial response with both Content-Digest and Repr-Digest

In the response message above, note that the Repr-Digest and Content-Digests are different. The Repr-Digest field-value is calculated across the entire JSON object {"hello": "world"}, and the field is

```
Repr-Digest: sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:
```

However, since the message content is constrained to bytes 1-7, the Content-Digest field-value is calculated over the byte sequence "hello", thus resulting in

```
NOTE: '\' line wrapping per RFC 8792

Content-Digest: \
  sha-256=:Wqdirjg/u3J688ejbUlApbjECpiUUtIwT8lY/z81Tno=:
```

B.4. Client and Server Provide Full Representation Data

The request contains a Repr-Digest field-value calculated on the enclosed representation. It also includes an Accept-Encoding: br header field that advertises the client supports Brotli encoding.

The response includes a Content-Encoding: br that indicates the selected representation is Brotli-encoded. The Repr-Digest field-value is therefore different compared to the request.

For presentation purposes, the response body is displayed as a Base64-encoded string because it contains non-printable characters.

```
PUT /items/123 HTTP/1.1
Host: foo.example
Content-Type: application/json
Accept-Encoding: br
Repr-Digest: sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:

{"hello": "world"}
```

Figure 16: PUT Request with Digest

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Location: /items/123
Content-Encoding: br
Content-Length: 22
Repr-Digest: sha-256=:4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=:

iwiAeyJoZWxsbyI6ICJ3b3JsZCJ9Aw==
```

Figure 17: Response with Digest of encoded response

B.5. Client Provides Full Representation Data, Server Provides No Representation Data

The request Repr-Digest field-value is calculated on the enclosed payload.

The response Repr-Digest field-value depends on the representation metadata header fields, including Content-Encoding: br even when the response does not contain content.

```
PUT /items/123 HTTP/1.1
Host: foo.example
Content-Type: application/json
Content-Length: 18
Accept-Encoding: br
Repr-Digest: sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:

{"hello": "world"}
```

```
HTTP/1.1 204 No Content
Content-Type: application/json
Content-Encoding: br
Repr-Digest: sha-256=:4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=:
```

Figure 18: Empty response with Digest

B.6. Client and Server Provide Full Representation Data

The response contains two digest values using different algorithms.

As the response body contains non-printable characters, it is displayed as a base64-encoded string.

```
PUT /items/123 HTTP/1.1
Host: foo.example
Content-Type: application/json
Accept-Encoding: br
Repr-Digest: sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:

{"hello": "world"}
```

Figure 19: PUT Request with Digest

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: br
Content-Location: /items/123
Repr-Digest: \
  sha-256=:4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=:,\
  sha-512=:pxo7aYzcGI88pnDnoSmAnaOEVys0MABhgvHY9+VI+ElE60jBCwnMPyA/\
  s3NF3ZO5oIWA7lf8ukk+5KJzm3p5og==:

iwiAeyJoZWxsbyI6ICJ3b3JsZCJ9Aw==
```

Figure 20: Response with Digest of Encoded Content

B.7. POST Response does not Reference the Request URI

The request Repr-Digest field-value is computed on the enclosed representation (see Section 2.1).

The representation enclosed in the response refers to the resource identified by Content-Location (see Section 6.4.2 of [SEMANTICS]). Repr-Digest is thus computed on the enclosed representation.

```
POST /books HTTP/1.1
Host: foo.example
Content-Type: application/json
Accept: application/json
Accept-Encoding: identity
Repr-Digest: sha-256=:bWopGGNiZtbVgHsG+I4knzfEJpmmmQHf7RHDXA3o1hQ=:

{"title": "New Title"}
```

Figure 21: POST Request with Digest

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Location: /books/123
Location: /books/123
Repr-Digest: sha-256=:yxOAqEeoj+reqygSIsLpT0LhumrNkIds5uLKtmdLyYE=:

{
  "id": "123",
  "title": "New Title"
}
```

Figure 22: Response with Digest of Resource

Note that a 204 No Content response without content but with the same Repr-Digest field-value would have been legitimate too. In that case, Content-Digest would have been computed on an empty content.

B.8. POST Response Describes the Request Status

The request Repr-Digest field-value is computed on the enclosed representation (see Section 2.1).

The representation enclosed in the response describes the status of the request, so Repr-Digest is computed on that enclosed representation.

Response Repr-Digest has no explicit relation with the resource referenced by Location.

```
POST /books HTTP/1.1
Host: foo.example
Content-Type: application/json
Accept: application/json
Accept-Encoding: identity
Repr-Digest: sha-256=:bWopGGNiZtbVgHsG+I4knzfEJpmmmQHf7RHDXA3o1hQ=:

{"title": "New Title"}
```

Figure 23: POST Request with Digest

```
HTTP/1.1 201 Created
Content-Type: application/json
Repr-Digest: sha-256=:2LBp5RKZGpsSNf8BPXlXrX4Td4Tf5R5bZ9z7kdi5VvY=:
Location: /books/123

{
  "status": "created",
  "id": "123",
  "ts": 1569327729,
  "instance": "/books/123"
}
```

Figure 24: Response with Digest of Representation

B.9. Digest with PATCH

This case is analogous to a POST request where the target resource reflects the effective request URI.

The PATCH request uses the application/merge-patch+json media type defined in [RFC7396].

Repr-Digest is calculated on the enclosed payload, which corresponds to the patch document.

The response Repr-Digest field-value is computed on the complete representation of the patched resource.

```
PATCH /books/123 HTTP/1.1
Host: foo.example
Content-Type: application/merge-patch+json
Accept: application/json
Accept-Encoding: identity
Repr-Digest: sha-256=:bWopGGNiZtbVgHsG+I4knzfEJpmmmQHf7RHDXA3o1hQ=:

{"title": "New Title"}
```

Figure 25: PATCH Request with Digest

```
HTTP/1.1 200 OK
Content-Type: application/json
Repr-Digest: sha-256=:yxOAqEeoj+reqygSIsLpT0LhumrNkIds5uLKtmdLyYE=:

{
  "id": "123",
  "title": "New Title"
}
```

Figure 26: Response with Digest of Representation

Note that a 204 No Content response without content but with the same Repr-Digest field-value would have been legitimate too.

B.10. Error responses

In error responses, the "representation data" does not necessarily refer to the target resource. Instead, it refers to the representation of the error.

In the following example, a client sends the same request from Figure 25 to patch the resource located at /books/123. However, the resource does not exist and the server generates a 404 response with a body that describes the error in accordance with [RFC7807].

The response Repr-Digest field-value is computed on this enclosed representation.

```
HTTP/1.1 404 Not Found
Content-Type: application/problem+json
Repr-Digest: sha-256=:KPqhVXAT25LLitV1w0O167unHmVQusu+fpxm65zAsvk=:

{
  "title": "Not Found",
  "detail": "Cannot PATCH a non-existent resource",
  "status": 404
}
```

Figure 27: Response with Digest of Error Representation

B.11. Use with Trailer Fields and Transfer Coding

An origin server sends Repr-Digest as trailer field, so it can calculate digest-value while streaming content and thus mitigate resource consumption. The Repr-Digest field-value is the same as in Appendix B.1 because Repr-Digest is designed to be independent from the use of one or more transfer codings (see Section 2).

```
GET /items/123 HTTP/1.1
Host: foo.example
```

Figure 28: GET Request

```
HTTP/1.1 200 OK
Content-Type: application/json
Transfer-Encoding: chunked
Trailer: Digest

8\r\n
{"hello"\r\n
8
: "world\r\n
2\r\n
"}\r\n
0\r\n
Repr-Digest: sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:
```

Figure 29: Chunked Response with Digest

Appendix C. Examples of Want-Repr-Digest Solicited Digest

The following examples demonstrate interactions where a client solicits a Repr-Digest using Want-Repr-Digest. The behavior of Content-Digest and Want-Content-Digest is identical.

Some examples include JSON objects in the content. For presentation purposes, objects that fit completely within the line-length limits are presented on a single line using compact notation with no leading space. Objects that would exceed line-length limits are presented across multiple lines (one line per key-value pair) with 2 spaced of leading indentation.

Checksum mechanisms described in this document are media-type agnostic and do not provide canonicalization algorithms for specific formats. Examples are calculated inclusive of any space.

C.1. Server Selects Client's Least Preferred Algorithm

The client requests a digest, preferring "sha". The server is free to reply with "sha-256" anyway.

```
GET /items/123 HTTP/1.1
Host: foo.example
Want-Repr-Digest: sha-256=3, sha=10
```

Figure 30: GET Request with Want-Repr-Digest

```
HTTP/1.1 200 OK
Content-Type: application/json
Repr-Digest: sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:

{"hello": "world"}
```

Figure 31: Response with Different Algorithm

C.2. Server Selects Algorithm Unsupported by Client

The client requests a "sha" digest because that is the only algorithm it supports. The server is not obliged to produce a response containing a "sha" digest, it instead uses a different algorithm.

```
GET /items/123 HTTP/1.1
Host: foo.example
Want-Repr-Digest: sha=10
```

Figure 32: GET Request with Want-Repr-Digest

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 200 OK
Content-Type: application/json
Repr-Digest: \
  sha-512=:WZDPaVn/7XgHaAy8pmojAkGWoRx2UFChF41A2svX+TaPm+AbwAgBWnrI\
  iYllu7BNNyealdVLvRwEmTHWXvJwew==:

{"hello": "world"}
```

Figure 33: Response with Unsupported Algorithm

C.3. Server Does Not Support Client Algorithm and Returns an Error

Appendix C.2 is an example where a server ignores the client's preferred digest algorithm. Alternatively a server can also reject the request and return an error.

In this example, the client requests a "sha" Repr-Digest, and the server returns an error with problem details [RFC7807] contained in the content. The problem details contain a list of the hashing algorithms that the server supports. This is purely an example, this specification does not define any format or requirements for such content.

```
GET /items/123 HTTP/1.1
Host: foo.example
Want-Repr-Digest: sha=10
```

Figure 34: GET Request with Want-Repr-Digest

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json

{
  "title": "Bad Request",
  "detail": "Supported hashing algorithms: sha-256, sha-512",
  "status": 400
}
```

Figure 35: Response advertising the supported algorithms

Acknowledgements

This document is based on ideas from [RFC3230], so thanks to J. Mogul and A. Van Hoff for their great work. The original idea of refreshing RFC3230 arose from an interesting discussion with M. Nottingham, J. Yasskin and M. Thomson when reviewing the MICE content coding.

Thanks to Julian Reschke for his valuable contributions to this document, and to the following contributors that have helped improve this specification by reporting bugs, asking smart questions, drafting or reviewing text, and evaluating open issues: Mike Bishop, Brian Campbell, Matthew Kerwin, James Manger, Tommy Pauly, Sean Turner, Justin Richer, and Erik Wilde.

Code Samples

RFC Editor: Please remove this section before publication.

How can I generate and validate the Repr-Digest values shown in the examples throughout this document?

The following python3 code can be used to generate digests for JSON objects using SHA algorithms for a range of encodings. Note that these are formatted as base64. This function could be adapted to other algorithms and should take into account their specific formatting rules.

```
import base64, json, hashlib, brotli, logging
log = logging.getLogger()

def encode_item(item, encoding=lambda x: x):
    indent = 2 if isinstance(item, dict) and len(item) > 1 else None
    json_bytes = json.dumps(item, indent=indent).encode()
    return encoding(json_bytes)

def digest_bytes(bytes_, algorithm=hashlib.sha256):
    checksum_bytes = algorithm(bytes_).digest()
    log.warning("Log bytes: \n[%r]", bytes_)
    return base64.encodebytes(checksum_bytes).strip()

def digest(item, encoding=lambda x: x, algorithm=hashlib.sha256):
    content_encoded = encode_item(item, encoding)
    return digest_bytes(content_encoded, algorithm)


item = {"hello": "world"}

print("Encoding | hashing algorithm | digest-value")
print("Identity | sha256 |", digest(item))
# Encoding | hashing algorithm | digest-value
# Identity | sha256 | X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=

print("Encoding | hashing algorithm | digest-value")
print("Brotli | sha256 |", digest(item, encoding=brotli.compress))
# Encoding | hashing algorithm | digest-value
# Brotli | sha256 | 4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=

print("Encoding | hashing algorithm | digest-value")
print("Identity | sha512 |", digest(item, algorithm=hashlib.sha512))
print("Brotli | sha512 |", digest(item, algorithm=hashlib.sha512,
                                    encoding=brotli.compress))
# Encoding | hashing algorithm | digest-value
# Identity | sha512 |b'WZDPaVn/7XgHaAy8pmojAkGWoRx2UFChF41A2svX+TaPm'
#                      '+AbwAgBWnrIiYllu7BNNyealdVLvRwEmTHWXvJwew=='
# Brotli | sha512 | b'pxo7aYzcGI88pnDnoSmAnaOEVys0MABhgvHY9+VI+ElE6'
#                    '0jBCwnMPyA/s3NF3ZO5oIWA7lf8ukk+5KJzm3p5og=='
```

Changes

RFC Editor: Please remove this section before publication.

Since draft-ietf-httpbis-digest-headers-06

* Remove id-sha-256 and id-sha-512 from the list of supported algorithms #855

Since draft-ietf-httpbis-digest-headers-05

*Reboot digest-algorithm values registry #1567

*Add Content-Digest #1542

*Remove SRI section #1478

Since draft-ietf-httpbis-digest-headers-04

*Improve SRI section #1354

*About duplicate digest-algorithms #1221

*Improve security considerations #852

*md5 and sha deprecation references #1392

*Obsolete 3230 #1395

*Editorial #1362

Since draft-ietf-httpbis-digest-headers-03

*Reference semantics-12

*Detail encryption quirks

*Details on Algorithm agility #1250

*Obsolete parameters #850

Since draft-ietf-httpbis-digest-headers-02

*Deprecate SHA-1 #1154

Avoid id- with encrypted content

*Digest is independent from MESSAGING and HTTP/1.1 is not normative #1215

*Identity is not a valid field value for content-encoding #1223

*Mention trailers #1157

*Reference httpbis-semantics #1156

*Add contentMD5 as an obsoleted digest-algorithm #1249

*Use lowercase digest-algorithms names in the doc and in the digest-algorithm IANA table.

Since draft-ietf-httpbis-digest-headers-01

*Digest of error responses is computed on the error representation-data #1004

*Effect of HTTP semantics on payload and message body moved to appendix #1122

*Editorial refactoring, moving headers sections up. #1109-#1112, #1116, #1117, #1122-#1124

Since draft-ietf-httpbis-digest-headers-00

*Align title with document name

Add id-sha- algorithm examples #880

*Reference [RFC6234] and [RFC3174] instead of FIPS-1

*Deprecate MD5

*Obsolete ADLER-32 but don't forbid it #828

*Update CRC32C value in IANA table #828

*Use when acting on resources (POST, PATCH) #853

*Added Relationship with SRI, draft Use Cases #868, #971

*Warn about the implications of Content-Location

Authors' Addresses

Roberto Polli
Team Digitale, Italian Government
Italy

Email: robipolli@gmail.com

Lucas Pardue
Cloudflare

Email: lucaspardue.24.7@gmail.com